

Cache Analysis of Non-uniform Distribution Sorting Algorithms

Naila Rahman

Department of Computer Science

University of Leicester

Leicester LE1 7RH, UK.

`naila@mcs.le.ac.uk`

Rajeev Raman*

Department of Computer Science,

University of Leicester,

Leicester LE1 7RH, UK.

`r.raman@mcs.le.ac.uk`

February 1, 2008

Abstract

We analyse the average-case cache performance of distribution sorting algorithms in the case when keys are independently but not necessarily uniformly distributed. The analysis is for both ‘in-place’ and ‘out-of-place’ distribution sorting algorithms and is more accurate than the analysis presented in [13]. In particular, this new analysis yields tighter upper and lower bounds when the keys are drawn from a uniform distribution. We use this analysis to tune the performance of the integer sorting algorithm MSB radix sort when it is used to sort independent uniform floating-point numbers (floats). Our tuned MSB radix sort algorithm comfortably outperforms a cache-tuned implementations of bucketsort [11] and Quicksort when sorting uniform floats from $[0, 1)$.

1 Introduction

Distribution sorting is a popular alternative to comparison-based sorting which involves placing n input keys into $k \leq n$ classes based on their value [6]. The classes are chosen so that all the keys in the i th class are smaller than all the keys in the $(i + 1)$ st class, for $i = 1, \dots, k - 1$, and furthermore, the class to which a key belongs can be computed in $O(1)$ time (e.g. if the keys are floats in the range $[a, b)$, we can calculate the class of a key x as $1 + \lfloor \frac{x-a}{b-a} \cdot k \rfloor$). Thus, the original sorting problem is reduced in linear time to the problem of sorting the keys in each class. A number of distribution sorting algorithms have been developed which run in linear (expected) time under some assumptions about the input keys, such as bucket sort and radix sort. Due to their poor *cache* utilisation, even good implementations—which minimise instruction counts—of these ‘linear-time’ algorithms fail to outperform general-purpose $O(n \log n)$ -time algorithms such as Quicksort or Mergesort on modern computers [8, 11].

Most algorithms are based upon the random-access machine model [1], which assumes that main memory is as fast as the CPU. However, in modern computers, main memory is typically one or two orders of magnitude slower than the CPU [4]. To mitigate this, one or more levels of *cache* are introduced between CPU and memory. A cache is a fast associative memory which holds the

*Supported in part by EPSRC grant GR/L92150

values of some main memory locations. If the CPU requests the contents of a memory location, and the value of that location is held in some level of cache (a cache *hit*), the CPU’s request is answered by the cache itself in typically 1-3 clock cycles; otherwise (a cache *miss*) it is answered by accessing main memory in typically 30-100 clock cycles. Since typical programs exhibit *locality of reference* [4], caches are often effective. However, algorithms such as distribution sort have poor locality of reference, and their performance can be greatly improved by optimising their cache behaviour. A number of papers have recently addressed this issue [7, 8, 11, 12, 9, 14], mostly in the context of sorting and related problems. There is also a large literature on algorithms specifically designed for hierarchical models of memory [15, 2], but there are some important differences between these models and ours (see [10] for a summary).

The cache performance of comparison-based sorting algorithms was studied in [8, 9, 14] and distribution sorting algorithms were considered in [7, 8, 11]. One pass of a distribution sort consists of a *count* phase where the number of keys in each class are determined, followed by a *permute* phase where the keys belonging to the same class are moved to consecutive locations in an array. We give an analysis of the cache behaviour of the permute phase, assuming the keys are independently drawn from a non-uniform distribution. In [13] we focused on ‘in-place’ permute, where the keys are rearranged without placing them first in an auxiliary array. In this paper we extend the analysis to ‘out-of-place’ permute. We model the above algorithms as probabilistic processes, and analyse the cache behaviour of these processes. For each process we give an exact expression for, as well as matching closed-form upper and lower bounds on, the number of misses.

In previous work on the cache analysis of distribution sorting, [7] have analysed the (somewhat easier) count phase for non-uniform keys, and [11] gave an empirical analysis of the permute phase for uniform keys. The process of *accessing multiple sequences* of memory locations, which arises in multi-way merge sort, was analysed previously by [9, 14]. The analysis in [9] assumes that accesses to the sequences are controlled by an adversary; our analysis demonstrates, among other things, that with uniform randomised accesses to the sequences, more sequences can be accessed optimally. In [14] a lower bound on cache misses is given for uniform randomised accesses; our lower bound is somewhat sharper. The analysis also improves upon the results in [13], by giving tighter upper and lower bounds when the keys are drawn from a uniform distribution.

In practice there are often cases when keys are not uniform (e.g., they may be normally distributed); our analysis can be used to tune distribution sort in these cases. We consider a different application here: sorting uniform floats using an *integer* sorting algorithm. It is well known that one can sort floats by sorting the bit-strings representing the floats, interpreting them as integers [4]. Since (simple) operations on integers are faster than operations on floats, this can improve performance; indeed, in [11] it was observed that an *ad hoc* implementation of the integer sorting algorithm *most-significant-bit first* radix sort (MSB radix sort) outperformed an optimised version of bucket sort on uniform floats. We observe that a uniform distribution on floating-point numbers induces a non-uniform distribution on the representing integers, and use our cache analysis to improve the performance of MSB radix sort on our machine. Our tuned ‘in-place’ MSB radix sort comfortably outperforms optimised implementations of other in-place or ‘in-place’ algorithms such as Quicksort or MPFlashsort [11], which is a cache-tuned version of bucket sort.

2 Cache preliminaries

This section introduces some terminology and notation regarding caches. The size of the cache is normally expressed in terms of two parameters, the *block size* (B) and the number of *cache blocks* (C). We consider main memory as being divided into equal-sized *blocks* consisting of B consecutively-numbered memory locations, with blocks starting at locations which are multiples of B . The cache is also divided into blocks of size B ; one cache block can hold the value of exactly one memory block. Data is moved to and from main memory only as blocks.

In a *direct-mapped* cache, the value of memory location x can only be stored in cache block $c = (x \text{ div } B) \bmod C$. If the CPU accesses location x and cache block c holds the values from x ’s block the access is a cache hit; otherwise it is a cache miss and the contents of the block containing x are copied into cache block c , *evicting* the current contents of cache block c . For our

```

Permute phase(out-of-place permutation)
1 for  $i := 0$  to  $n - 1$  do
     $key := DATA[i]$ ;
     $x := \text{classify}(key)$ 
     $idx := COUNT[x]$ ;
     $COUNT[x]++$ ;
     $DEST[idx] := key$ ;

```

Figure 1: Permute phase for an ‘out-of-place’ permutation in a generic distribution sorting algorithm. `DATA` holds the input keys. `COUNT` and `DEST` are auxiliary arrays.

purposes, cache misses can be classified into *compulsory misses*, which occur when a memory block is accessed for the first time, *capacity misses*, which occurs on an access to a memory block that was previously evicted because the cache could not hold all the blocks being actively accessed, and *conflict misses*, which happen when a block is evicted from cache because another memory block that mapped to the same cache block was accessed.

3 Distribution sorting

As noted in the introduction, a distribution pass has two main phases, a *count* phase and a *permute* phase, and our focus here is on the latter.

While describing this algorithm, the term *data* array refers to the array holding the input keys, and the term *count* refers to an auxiliary array used by these algorithms. Each pass consists of two main phases, a *count* phase followed by a *permute* phase.

The count phase counts for class $1 \leq i \leq k - 1$, the total number of keys in classes $0, \dots, i - 1$. For class $i = 0$ this cumulative count is 0. Ladner et al [7] give an analysis of the count phase of distribution sorting on a direct-mapped cache for uniformly and randomly distributed keys.

There are two main variants of the permute phase, in the first variant keys are permuted from the data array to the auxiliary *destination* array, this is called an *out-of-place permutation*. In the second variant keys in the data array are permuted within the data array, this is called an *in-place permutation*.

3.1 Permute phase

The permute phase uses the cumulative count of keys generated during the count phase, to permute the keys to their respective classes. We now describe the two variants of the permute phase. In the description below it is assumed that k has been appropriately initialised, and that the function `classify` maps a key to a class numbered $\{0, \dots, k - 1\}$ in $O(1)$ time.

3.1.1 Out-of-place permute

During an out-of-place permute, for any class j , unless all elements of that class have already been moved, `COUNT[j]` points to the leftmost (lowest-numbered) available location for an element of class j in an n element auxiliary array, `DEST`. Figure 1 shows the pseudo-code for out-of-place permutation. In Step 1, for each element in `DATA`: we determine its class; using the count array we determine the next available location for this key in the `DEST` array; we increment the count array, thus setting the location for the next key of the same class; finally we move the key to its location in `DEST`. Since each step takes constant time, this out-of-place permutation takes $O(n)$ time whenever $k \leq n$.

```

Permute phase(in-place permutation)
1 leader :=  $n - 1$ ;
2 idx := leader; key := DATA[idx];
3.1  $x$  := classify(key);
3.2 idx := COUNT[ $x$ ];
3.3 COUNT[ $x$ ]++;
3.4 swap key and DATA[idx];
3.5 if  $idx \neq leader$  repeat 3.1;
4 while ( $x > 0 \wedge COUNT[x - 1] \geq START[x]$ )
     $x--$ ;
5 if ( $x > 0$ ) leader := START[ $x$ ] - 1;
   go to 2;

```

Figure 2: Permute phase for an ‘in-place’ permutation in a generic distribution sorting algorithm. DATA holds the input keys. COUNT and START are auxiliary arrays. After the count phase, COUNT is copied into START.

3.1.2 In-place Permute

The in-place permutation strategy described here is similar to that described by Knuth [6, Soln 5.2-13]. Before an in-place permute phase begins, a copy of the count array is made in a k element auxiliary *start* array. During the permute phase, for any class j , an invariant is that locations $START[j], START[j] + 1, \dots, COUNT[j] - 1$ contain elements of class j , i.e. $COUNT[j]$ points to the leftmost (lowest-numbered) available location for an element of class j . Thus, for $j = 0, \dots, k - 2$, all elements of class j have been permuted if $COUNT[j] \geq START[j + 1]$, and such a class will be called *complete* in what follows. Class $k - 1$ is complete when $COUNT[k - 1] \geq n$. Figure 2 shows the pseudo-code for in-place permutation. We now describe this permutation, which consists of two main activities: *cycle following* and *cycle leader finding*. In cycle following, keys are moved to their final destinations in the data array along a cycle in the permutation (Steps 2 and 3). Once a cycle is completed, we move to cycle leader finding, where we find the ‘leader’ (index of the rightmost element) of the next cycle (Steps 1, 4 and 5). A cycle leader is simply the rightmost location of the highest-numbered incomplete class. By the definition of a complete class, initially the leader must be position $n - 1$. In more detail, the steps are as follows:

- In Step 1 $n - 1$ is selected as the first cycle leader.
- In Step 2 the key at the leader’s position is copied into the variable *key*, thus leaving a ‘hole’ in the leader’s position.
- In Steps 3.1-3.5 the key *key* is swapped with the key at *key*’s final position. If *key* ‘fills the hole’, the cycle is complete, otherwise we repeat these steps.
- In Step 4 the algorithm searches for a new cycle leader. Suppose the leader of the cycle which just completed was the last location of class j . When this cycle ends, class j must also be complete, as a key of class j has been moved into the last location of class j . Note that classes $j + 1, j + 2, \dots$ must already have been complete when the leader of this cycle was found. Note that the program variable x has value j at the end of this cycle, so the search for the next leader begins with class $j - 1$, counting down (Step 4).
- In Step 5 we check to see if all classes have completed and terminate if this is the case.

Clearly the in-place permutation in one pass of distribution sorting takes $O(n)$ time whenever $k \leq n$.

4 Cache analysis

We now analyse cache misses in a direct-mapped cache during the permute phase of distribution sorting when the keys are independently drawn from a non-uniform random distribution. In the permute phase of distribution sorting, when a key is moved to its destination, the algorithms described in Section 3 access any one of k elements in the **COUNT** array and any one of k locations in the **DATA** or **DEST** arrays, depending on whether the permutation is in-place or out-of-place. The actual locations accessed are dependent on the value of the permuted key, so, if the keys are independently and randomly distributed then, for every key permuted there are two random accesses to memory, one in the count array and one in **DATA** or **DEST**. These random accesses can potentially lead to a large number of cache conflict misses.

Our approach is to define two continuous processes which model in-place and out-of-place permutations. Process “in-place” models an in-place permutation and is shown in Figure 3, and Process “out-of-place” models an out-of-place permutation and is shown in Figure 4. Each round of a process models the permutation of a key to its destination, and we analyse the expected number of cache misses in n rounds of these processes. Our precise equations are difficult to compute so we also give closed-form upper and lower bounds on these precise equations. We use our results for in-place permutations to get upper and lower bounds on the expected number of cache misses in a process which models accesses to multiple sequences.

The assumptions in the processes mean that we have to access at least n distinct locations in memory, which requires $\Omega(n/B)$ cache misses. In the analysis, we will say that a process is optimal if it incurs $O(n/B)$ cache misses. In distribution sorting, the larger the value of k , the fewer the number of passes over the data, hence the fewer the capacity misses. As we will see, if k is too large, then there can be a large number of conflict misses. The aim of the analysis is to determine the largest value of k , for a particular distribution of keys, such that there are $O(n/B)$ misses in one pass of distribution sorting.

4.1 Processes

We now give the two processes which model the distributing of keys drawn independently and randomly from a non-uniform distribution into k classes.

4.1.1 Process to model an in-place permutation

Let k be an integer, $2 \leq k \leq CB$. We are given k probabilities p_1, \dots, p_k , such that $\sum_{i=1}^k p_i = 1$. The process maintains k pointers D_1, \dots, D_k , and there are also k consecutive ‘count array’ locations, $\mathcal{C} = c_1, \dots, c_k$. The process (henceforth called *Process “in-place”*) executes a sequence of *rounds*, where each round consists in performing steps 1-3 below:

Process “in-place”
1. Pick an integer x from $\{1, \dots, k\}$ such that $\Pr[x = i] = p_i$, independently of all previous picks.
2. Access the location c_x .
3. Access the location pointed to by D_x , increment D_x by 1.

We denote the locations accessed by the pointer D_i by $d_{i,1}, d_{i,2}, \dots$, for $i = 1, \dots, k$. We assume that:

- (a) the start position of each pointer is uniformly and independently distributed over the cache, i.e., for each i , $d_{i,1} \bmod BC$ is uniformly and independently distributed over $\{0, \dots, BC - 1\}$,
- (b) during the process, the pointers traverse sequences of memory locations which are disjoint from each other and from \mathcal{C} ,

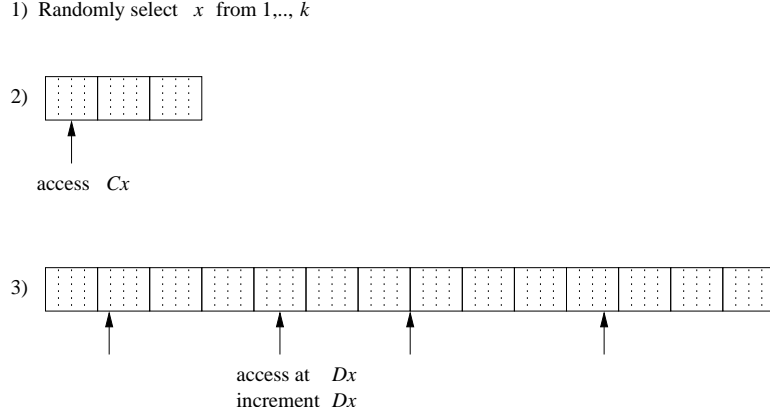


Figure 3: Process “Inplace”.

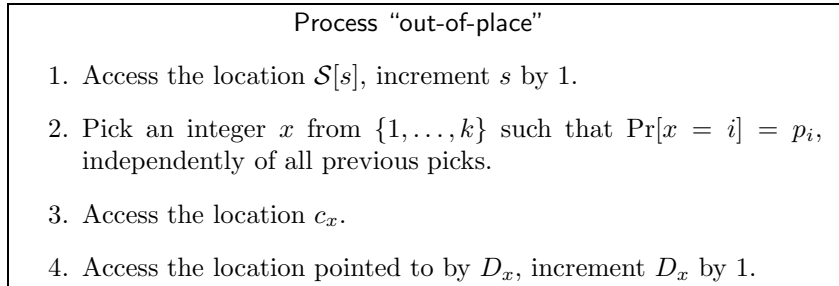
(c) c_1 is located on an aligned block boundary, i.e., $c_1 \bmod B = 0$,

(d) the pointers D_i , for $i = 1, \dots, k$, are in separate memory blocks.

Assuming that the cache is initially empty, the objective is to determine the expected number of cache misses incurred by the above process over n rounds, with the expectation taken over the random choices in Step 1 as well as the starting positions of the pointers.

4.1.2 Process to model an out-of-place permutation

This process is like Process “in-place”, but it is augmented with accesses to a sequence of consecutive locations in a source array, \mathcal{S} , determined by an index s . The process, henceforth called *Process “out-of-place”*, executes a sequence of *rounds*, where each round consists in performing steps 1-4 below:



We make assumptions (a), (c), and (d) from Process “in-place”, assumption (b) is modified as below and we add a further assumption:

(b) during the process, the pointers traverse sequences of memory locations which are disjoint from each other, from \mathcal{C} and from \mathcal{S} .

Assuming that the cache is initially empty, again the objective is to determine the expected number of cache misses incurred by the above process over n rounds, with the expectation taken over the random choices in Step 2 as well as the starting positions of the pointers.

4.2 Preliminaries

We now introduce some notation that will be used for the analysis. We use k to denote the number of classes that the keys will be distributed into, and throughout the analysis we assume that B divides k . Assume that we are given a set of k probabilities p_1, \dots, p_k , such that $\sum_{i=1}^k p_i = 1$. The

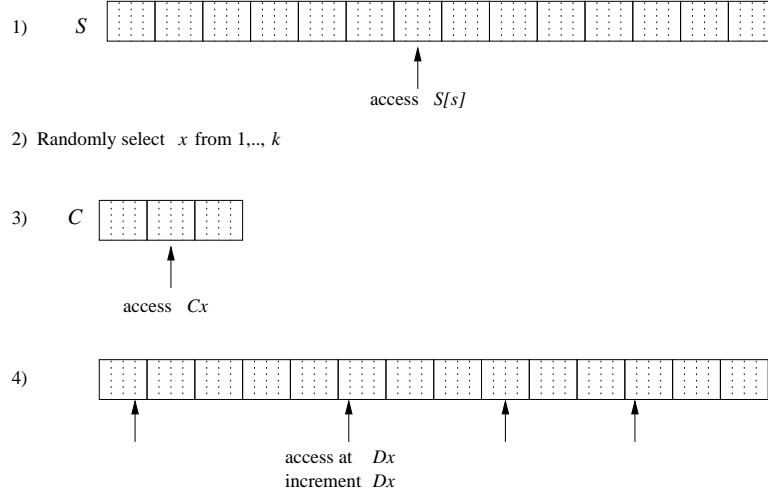


Figure 4: Process “Out-of-place”.

expected value of a function f of a random variable X is denoted as $E[f(X)]$. When we wish to make explicit the distribution D from which the random variable is drawn, we will use the notation $E_{X \sim D}[f(X)]$. All vectors have dimension k (the number of classes) unless stated otherwise, and we denote the components of a vector \bar{x} by x_1, x_2, \dots, x_k . We now define some probabilities:

- (i) For all $i \in \{1, \dots, k/B\}$, $P_i = \sum_{l=(i-1)B+1}^{iB} p_l$.
- (ii) For all $i \in \{1, \dots, k\}$, we denote by \bar{a}^i the following vector: $a_j^i = 0$ if $i \neq j$, and $a_j^i = p_j/(1 - p_i)$ otherwise and by \bar{b}^i the following vector: $b_j^i = 0$ if $(i-1)B + 1 \leq j \leq iB$, and $b_j^i = p_j/(1 - P_i)$ otherwise. (Note that $\sum_j a_j^i = \sum_j b_j^i = 1$).

Let $m \geq 0$ be an integer and \bar{q} be a vector of non-negative reals such that $\sum_i q_i = 1$. We denote by $\varphi(m, \bar{q})$ the probability distribution on the number of balls in each of k bins, when m balls are independently put into these bins, and a ball goes in bin i with probability q_i , for $i \in \{1, \dots, k\}$. Thus, $\varphi(m, \bar{q})$ is a distribution on vectors of non-negative integers. If $\bar{\mu}$ is drawn from $\varphi(m, \bar{q})$, then:

$$\Pr[\mu_1 = m_1, \dots, \mu_k = m_k] = \left(\prod_{j=1}^k q_j^{m_j} \right) m! / \prod_{j=1}^k m_j! \quad (1)$$

whenever $\sum_{i=1}^k m_i = m$; all other vectors have zero probability¹. We now define functions $f(x)$ for $x \geq 0$ and $g(\bar{m})$ for a vector \bar{m} of non-negative integers:

$$f(x) = \begin{cases} 1 & \text{if } x = 0, \\ 1 - \frac{x+B-1}{BC} & \text{if } 0 < x \leq BC - B + 1, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

$$g(\bar{m}) = \frac{1}{C} \sum_{i=1}^{k/B} \min\{1, \sum_{l=(i-1)B+1}^{iB} m_l\}. \quad (3)$$

We now set out some propositions that are used in the proofs.

Proposition 1 *For all real numbers $x_i, i = 1, \dots, k$, such that $|x_i| \leq 1$ we have that:*

$$\prod_{i=0}^k (1 - x_i) \geq 1 - \sum_{i=0}^k x_i.$$

¹We take $0^0 = 1$ in Eq. 1.

Proposition 2 (a) For all real numbers x , such that $|x| < 1$ we have that:

$$\sum_{m=0}^{\infty} x^m = \frac{1}{1-x}.$$

(b) For all real numbers x , such that $|x| < 1$ we have that:

$$\sum_{m=0}^{\infty} mx^m = \frac{x}{(1-x)^2}.$$

(c) For all real numbers x , such that $0 < x < 2$, we have that:

$$\sum_{m=0}^{\infty} x(1-x)^m m = \frac{1}{x} - 1.$$

PROOF. Proposition 2(a) is the standard summation for an infinite decreasing geometric series. We obtain Proposition 2(b) by differentiating both sides of the equation in Proposition 2(a). Proposition 2(c) is obtained using Proposition 2(b) and is the expected value of the geometric distribution multiplied by $1-x$. \square

Proposition 3 For all real numbers p and q such that $0 < p-q < 2$, we have that:

$$\sum_{m=0}^{\infty} p(1-p)^m \left(1 - \frac{q}{1-p}\right)^m = \frac{p}{p+q}.$$

PROOF. Since $(1-p)\left(1 - \frac{q}{1-p}\right) = 1-p-q$, using Proposition 2(a) we get that:

$$\sum_{m=0}^{\infty} p(1-p)^m \left(1 - \frac{q}{1-p}\right)^m = \sum_{m=0}^{\infty} p(1-p-q)^m = \frac{p}{p+q}.$$

\square

Proposition 4 (a) For all real numbers x , we have that:

$$e^{-x} \geq 1-x.$$

(b) For all real numbers $x \geq 0$, we have that:

$$e^{-x} \leq 1-x + \frac{x^2}{2}.$$

(c) For all real numbers $x_i, i = 1, \dots, k$, such that $x_i \leq 1$ we have that:

$$\prod (1-x_i) \leq 1 - \sum x_i + \sum \frac{x_i^2}{2}.$$

PROOF. Propositions 4(a) and 4(b) are from Taylor's series. For Propositions 4(c) we use Proposition 1. \square

Proposition 5 For all real numbers x and y , such that $x \leq 1$ and $y \geq 0$, we have that:

$$e^{-xy} \geq (1-x)^y.$$

PROOF. This proposition is proved using Proposition 4(a). \square

Proposition 6 (a) For all real numbers x and p and integer y , such that $0 < p \leq 1$, $y \geq 0$ and $x(1/p + y) = O(1)$, we have that:

$$\sum_{m=0}^y p(1-p)^m mx = x \left(\frac{1}{p} - 1 \right) - O(e^{-py}).$$

(b) For all real numbers x and p and integer y , such that $0 < p \leq 1$, $y \geq 0$ and $x = O(1)$, we have that:

$$\sum_{m=0}^y p(1-p)^m x = x - O(e^{-py}).$$

(c) For all real numbers x , p and q and integer y , such that $0 < p - q < 2$, $y \geq 0$ and $\frac{xp}{p+q} = O(1)$, we have that:

$$\sum_{m=0}^y p(1-p)^m x = \frac{xp}{p+q} - O(e^{-(p+q)y}).$$

(d) For all real numbers m , x , p and q and integer y , such that $0 < p - q < 2$, $y \geq 0$ and $\frac{xp}{p+q}(\frac{1-p-q}{p+q} + y + 1) = O(1)$, we have that:

$$\sum_{m=0}^y p(1-p)^m x = \frac{xp(1-p-q)}{(p+q)^2} - O(e^{-(p+q)y}).$$

Note that we are misusing the O notation here to hide constant factors that are independent of the variables in the equations.

PROOF. Using Proposition 2(c) and Proposition 5, Proposition 6(a) is proved as follows:

$$\begin{aligned} \sum_{m=0}^y p(1-p)^m mx &= \sum_{m=0}^{\infty} p(1-p)^m mx - (1-p)^{y+1} \sum_{m=0}^{\infty} p(1-p)^m (m+y+1)x \\ &= x \left(\frac{1}{p} - 1 \right) - (1-p)^{y+1} x \left(\frac{1}{p} + y \right) \\ &= x \left(\frac{1}{p} - 1 \right) - O(e^{-py}). \end{aligned}$$

The proofs of Propositions 6(b), 6(c) and 6(d) are now trivial. \square

The vector of random variables $X = (X_1, \dots, X_n)$, is *negatively associated* [5] if for every two disjoint index sets, $I, J \subset [n]$,

$$\mathbb{E}[f(X_i, i \in I)g(X_j, j \in J)] \leq \mathbb{E}[f(X_i, i \in I)]\mathbb{E}[g(X_j, j \in J)]$$

for all functions $f : \mathbb{R}^{|I|} \rightarrow \mathbb{R}$ and $g : \mathbb{R}^{|J|} \rightarrow \mathbb{R}$ that are both non-decreasing or non-increasing.

Proposition 7 If the random variables X_1, \dots, X_k are negatively associated, then for any non-decreasing function $f_i, i \in [k]$, we have that:

$$\mathbb{E}[\prod_{i=1}^k f_i(X_i)] \leq \prod_{i=1}^k \mathbb{E}[f_i(X_i)].$$

PROOF. The proof follows directly from the definition of negatively associated variables. \square

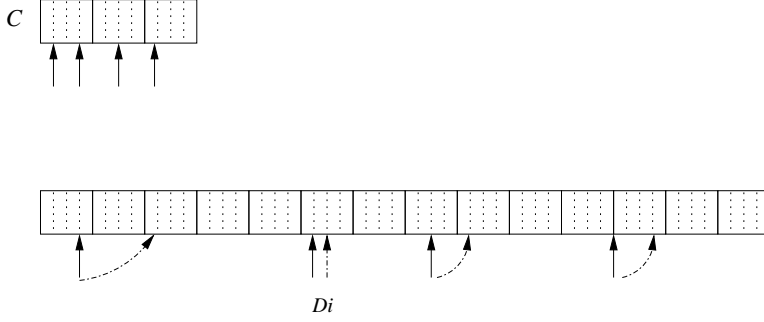


Figure 5: m rounds of Process “in-place”. Between two accesses to D_i , there are m accesses to “other” pointers, and $m + 1$ accesses to \mathcal{C} .

4.3 Cache Analysis of In-place Permutation

In this section we analyse the cache misses in a direct mapped cache during n rounds of Process “in-place”, introduced in Section 4.1.1. We derive a precise equation for the expected number of cache misses and then give closed form upper and lower bounds on this equation. We then derive upper and lower bounds assuming the keys are drawn independently from a uniform distribution.

4.3.1 Average case analysis

We start by proving a theorem for the expected number of cache misses during n rounds of Process “in-place”.

Theorem 1 *The expected number X of cache misses in n rounds of Process “in-place” satisfies $n(p_c + p_d) \leq X \leq n(p_c + p_d) + k(1 + 1/B)$, where:*

$$\begin{aligned}
 p_c &= \sum_{i=1}^{k/B} P_i \left(1 - \sum_{m=0}^{\infty} P_i (1 - P_i)^m \mathbb{E}_{\bar{\nu} \sim \varphi(m, \bar{b}_i)} \left[\prod_{j=1}^k f(\nu_j) \right] \right) \text{ and} \\
 p_d &= \frac{1}{B} + \\
 &\quad \frac{B-1}{B} \sum_{i=1}^k p_i \left(1 - \sum_{m=0}^{\infty} p_i (1 - p_i)^m \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} \left[(1 - g(\bar{\mu})) \prod_{j=1}^k f(\mu_j) \right] \right).
 \end{aligned}$$

PROOF. We first analyse the miss rates for accesses to pointers D_1, \dots, D_k . Fix an i , $1 \leq i \leq k$ and a $z \geq 1$. Let μ be the random variable which denotes the number of rounds between accesses to locations $d_{i,z}$ and $d_{i,z+1}$ ($\mu = 0$ if these locations are accessed in consecutive rounds). Figure 5 shows the other memory accesses between accesses z and $z + 1$ to D_i . Clearly, $\Pr[\mu = m] = p_i(1 - p_i)^m$, for $m = 0, 1, \dots$. Let X_i denote the event that none of the memory accesses in these μ rounds accesses the cache block to which $d_{i,z}$ is mapped. We now fix an integer $m \geq 0$ and calculate $\Pr[X_i | \mu = m]$. Let $\bar{\mu}$ be a vector of random variables such that for $1 \leq j \leq k$, μ_j is the random variable which denotes the number of accesses to D_j in these m rounds. Clearly $\bar{\mu}$ is drawn from $\varphi(m, \bar{a}_i)$ (note that D_i is not accessed in these m rounds by definition).

Fix any vector \bar{m} , such that $\Pr[\bar{\mu} = \bar{m}] \neq 0$, and let μ_j be the number of accesses to pointer D_j in these m rounds. Since m_i must be zero, $f(m_i) = 1$, and for $j \neq i$, $f(m_j)$ is the probability that none of the m_j locations accessed by D_j in these m rounds is mapped to the same cache block as location $d_{i,z}$ [9, 14]. Similarly $g(\bar{m}) \cdot C$ is the number of count blocks accessed in these rounds, and so $1 - g(\bar{m})$ is the probability that the cache block containing $d_{i,z}$ does not conflict with the blocks from \mathcal{C} which were accessed in these rounds. As the latter probability is determined by the starting location of sequence i and the former probabilities by the starting location of sequences $j, j \neq i$, we conclude that for a given configuration \bar{m} of accesses, the probability that the cache

block containing $d_{i,z}$ is not accessed in these m rounds is $(1 - g(\bar{m})) \prod_{j=1}^k f(m_j)$. Averaging over all configurations \bar{m} , we get that

$$\Pr[X_i \mid \mu = m] = E_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[(1 - g(\bar{\mu})) \prod_{j=1}^k f(\mu_j)]. \quad (4)$$

Finally we get,

$$\begin{aligned} \Pr[X_i] &= \sum_{m=0}^{\infty} \Pr[\mu = m] \Pr[X_i \mid \mu = m] \\ &= \sum_{m=0}^{\infty} p_i (1 - p_i)^m E_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} \left[(1 - g(\bar{\mu})) \prod_{j=1}^k f(\mu_j) \right]. \end{aligned} \quad (5)$$

If $d_{i,z}$ is at a cache block boundary or if X_i does not occur given that $d_{i,z}$ is not at a cache block boundary ($\Pr[X_i]$ does not change under this condition), then a cache miss will occur. The first access to a pointer is a cache miss. So other than for the first access, the probability p_d of a cache miss for a pointer access is:

$$p_d = \frac{1}{B} + \frac{B-1}{B} \sum_{i=1}^k p_i (1 - \Pr[X_i]). \quad (6)$$

Including the first access misses, the expected number of cache misses for pointer accesses is at most

$$\sum_{i=1}^k 1 + (np_i - 1) \left(\left(\frac{B-1}{B} (1 - \Pr[X_i]) \right) + \frac{1}{B} \right) \leq np_d + k. \quad (7)$$

We now consider the probability of a cache miss for an access to a count array location. It is convenient to partition \mathcal{C} into count blocks of B locations each, where the i -th count block consists of the locations $c_{(i-1)B+1}, \dots, c_{iB}$, for $i = 1, \dots, k/B$. So P_i is the probability of access to the i -th block. We fix an $i \in \{1, \dots, k/B\}$ and a $z \geq 1$. Let ν be the random variable that denotes the number of rounds between the z -th and $(z+1)$ -st accesses to the i -th count block. We have that $\Pr[\nu = m] = P_i (1 - P_i)^m$, for $m = 0, 1, \dots$. Let Y_i denote the event that none of the memory accesses in these m rounds accesses the cache block to which the i -th count block is mapped.

We now fix an integer $m \geq 0$ and calculate $\Pr[Y_i \mid \nu = m]$. Let $\bar{\nu}$ be a vector of random variables such that for $1 \leq j \leq k$, ν_j is the random variable which denotes the number of accesses to D_j in these m rounds. Given that $k \leq BC$ and assumption (c) mean that two blocks from \mathcal{C} cannot conflict with each other. As the pointers $D_{(i-1)B+1}, \dots, D_{iB}$ will not be accessed between two successive accesses to count block i , the probability of accessing pointer D_j is given by b_j^i and $\varphi(m, \bar{b}_i)$ is the distribution for $\bar{\nu}$. Arguing as above:

$$\begin{aligned} \Pr[Y_i] &= \sum_{m=0}^{\infty} \Pr[\nu = m] \Pr[Y_i \mid \nu = m] \\ &= \sum_{m=0}^{\infty} P_i (1 - P_i)^m E_{\bar{\nu} \sim \varphi(m, \bar{b}_i)} \left[\prod_{j=1}^k f(\nu_j) \right]. \end{aligned} \quad (8)$$

The first access to a count array block is a cache miss, for all other accesses there is a cache miss if event Y_i does not occur. So other than for the first access, the probability p_c of a cache miss for a count array access is:

$$p_c = \sum_{i=1}^{k/B} P_i (1 - \Pr[Y_i]). \quad (9)$$

Including the first access misses, the expected number of cache misses for count array accesses is at most

$$\sum_{i=1}^{k/B} 1 + (nP_i - 1)(1 - \Pr[Y_i]) \leq np_c + k/B. \quad (10)$$

Plugging in the values from Eq. 5 into Eq. 7 and from Eq. 8 into Eq. 10 we get the upper bound on X , the expected number of cache misses in the processes. The lower bound in Theorem 1 is obvious. \square

4.3.2 Upper bound

We now prove a theorem on the upper bound to the expected number of cache misses during n rounds of Process “in-place”.

Theorem 2 *The expected number of cache misses in n rounds of Process “in-place” is at most $n(p_d + p_c) + k(1 + 1/B)$, where:*

$$\begin{aligned} p_d &\leq \frac{1}{B} + \frac{k}{BC} + \frac{B-1}{BC} \sum_{i=1}^k \left(\sum_{j=1}^{k/B} \frac{p_i P_j}{p_i + P_j} + \frac{B-1}{B} \sum_{j=1}^k \frac{p_i p_j}{p_i + p_j} \right), \\ p_c &\leq \frac{k}{B^2 C} + \frac{B-1}{BC} \sum_{i=1}^{k/B} \sum_{j=1}^k \frac{P_i p_j}{P_i + p_j}. \end{aligned}$$

PROOF. In the proof we derive lower bounds for $\Pr[X_i]$ and $\Pr[Y_i]$ and use these to derive the upper bounds on p_d and p_c .

Again, we consider a fixed i and consider the event X_i defined in the proof of Theorem 1. We now obtain a lower bound on $\Pr[X_i]$.

Lower bound on $\Pr[X_i]$

Letting $\Gamma(x) = 1 - f(x)$ and using Proposition 1 we can rewrite Eq. 5 as:

$$\Pr[X_i] \geq \sum_{m=0}^{\infty} \Pr[\mu = m] \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} \left[1 - g(\bar{\mu}) - \sum_{j=1}^k \Gamma(\mu_j) \right]. \quad (11)$$

We know that the j -th count block contributes $1/C$ to $g(\bar{\mu})$ if there is an access to that block and $\Pr[j\text{-th count block accessed} | \mu = m] = 1 - (1 - c_j^i)^m$, where $c_j^i = \frac{P_j}{1 - p_i}$. So we have that,

$$\mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [g(\bar{\mu})] = \sum_{j=1}^{k/B} \frac{1}{C} (1 - (1 - c_j^i)^m),$$

and we get,

$$\begin{aligned} \sum_{m=0}^{\infty} \Pr[\mu = m] \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [g(\bar{\mu})] &= \sum_{m=0}^{\infty} p_i (1 - p_i)^m \sum_{j=1}^{k/B} \frac{1}{C} (1 - (1 - c_j^i)^m) \\ &= \frac{1}{C} \sum_{j=1}^{k/B} \sum_{m=0}^{\infty} p_i (1 - p_i)^m (1 - (1 - c_j^i)^m), \end{aligned}$$

and using Proposition 3 we get,

$$\sum_{m=0}^{\infty} \Pr[\mu = m] \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [g(\bar{\mu})] = \frac{1}{C} \sum_{j=1}^{k/B} \frac{P_j}{p_i + P_j}. \quad (12)$$

We now evaluate

$$\sum_{m=0}^{\infty} \Pr[\mu = m] \sum_{j=1}^k \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)].$$

Our approach is to first fix j and evaluate $\mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)]$. For $m \leq BC$, we know that

$$\mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)] = \sum_{l=0}^m \Pr[\mu_j = l] \frac{l+B-1}{BC} - \Pr[\mu_j = 0] \frac{B-1}{BC}.$$

The last term is due to the fact that $\Gamma(x)$ is discontinuous and $\Gamma(0) = 0$. Similarly for $m > BC$ we know that

$$\begin{aligned} \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)] &= \sum_{l=0}^m \Pr[\mu_j = l] \frac{l+B-1}{BC} - \Pr[\mu_j = 0] \frac{B-1}{BC} \\ &\quad - \sum_{l=BC-B+1}^m \Pr[\mu_j = l] \left(\frac{l+B-1}{BC} - 1 \right). \end{aligned}$$

The last term is due to the fact that $\Gamma(x) = 1$ for $x \geq BC - B + 1$. If we drop this last term when $m > BC$, we get that for all m

$$\mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)] \leq \frac{1}{BC} \left[\sum_{l=0}^m \Pr[\mu_j = l] l + (B-1)(1 - \Pr[\mu_j = 0]) \right].$$

The summation term is the expected value of the random variable with the binomial distribution $b(l; m, a_j^i)$. So we get that

$$\mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)] \leq \frac{1}{BC} \left[m a_j^i + (B-1) \left(1 - (1 - a_j^i)^m \right) \right]. \quad (13)$$

We now evaluate $\sum_{m=0}^{\infty} \Pr[\mu = m] \sum_{j=1}^k \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)]$ as

$$\begin{aligned} &\sum_{m=0}^{\infty} \Pr[\mu = m] \sum_{j=1}^k \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)] \\ &\leq \sum_{m=0}^{\infty} \Pr[\mu = m] \sum_{j=1}^k \frac{1}{BC} \left[m a_j^i + (B-1) \left(1 - (1 - a_j^i)^m \right) \right]. \end{aligned}$$

Since $\sum_{j=1}^k m a_j^i = m$, we get $\sum_{m=0}^{\infty} \Pr[\mu = m] \sum_{j=1}^k m a_j^i = \frac{1}{p_i} - 1$ by an application of Proposition 2(c). By applying Proposition 3 we get that $\sum_{j=1}^k \sum_{m=0}^{\infty} \Pr[\mu = m] (B-1)(1 - (1 - a_j^i)^m) = (B-1) \sum_{j=1}^k \frac{p_j}{p_i + p_j}$. So we get:

$$\sum_{m=0}^{\infty} \Pr[\mu = m] \sum_{j=1}^k \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu_j)] \leq \frac{1}{BC} \left(\frac{1}{p_i} + (B-1) \sum_{j=1}^k \frac{p_j}{p_i + p_j} \right). \quad (14)$$

Substituting Eq. 12 and 14 in Eq. 11 we obtain the following lower bound for $\Pr[X_i]$

$$\Pr[X_i] \geq 1 - \frac{1}{C} \sum_{j=1}^{k/B} \frac{P_j}{p_i + P_j} - \frac{1}{BC} \left(\frac{1}{p_i} + (B-1) \sum_{j=1}^k \frac{p_j}{p_i + p_j} \right). \quad (15)$$

Upper bound on p_d

Finally, substituting $\Pr[X_i]$ from Eq. 15 in Eq. 6 we get:

$$\begin{aligned}
p_d &\leq \frac{1}{B} + \\
&\quad \frac{B-1}{B} \sum_{i=1}^k p_i \left(\frac{1}{C} \sum_{j=1}^{k/B} \frac{P_j}{p_i + P_j} + \frac{1}{BC} \left(\frac{1}{p_i} + (B-1) \sum_{j=1}^k \frac{p_j}{p_i + p_j} \right) \right) \\
&= \frac{1}{B} + \frac{(B-1)k}{B^2C} + \frac{B-1}{BC} \sum_{i=1}^k \sum_{j=1}^{k/B} \frac{p_i P_j}{p_i + P_j} + \frac{(B-1)^2}{B^2C} \sum_{i=1}^k \sum_{j=1}^k \frac{p_i p_j}{p_i + p_j} \\
&\leq \frac{1}{B} + \frac{k}{BC} + \frac{B-1}{BC} \sum_{i=1}^k \left(\sum_{j=1}^{k/B} \frac{p_i P_j}{p_i + P_j} + \frac{B-1}{B} \sum_{j=1}^k \frac{p_i p_j}{p_i + p_j} \right).
\end{aligned}$$

We can evaluate p_c using a very similar approach, as sketched out now. We again consider a fixed i and consider the event Y_i defined in the proof of Theorem 1. We now obtain a lower bound on $\Pr[Y_i]$.

Lower bound on $\Pr[Y_i]$

Again letting $\Gamma(x) = 1 - f(x)$ and using Proposition 1, we can rewrite Eq. 8 as:

$$\Pr[Y_i] \geq \sum_{m=0}^{\infty} \Pr[\nu = m] \mathbb{E}_{\bar{\nu} \sim \varphi(m, \bar{b}_i)} \left[1 - \sum_{j=1}^k \Gamma(\nu_j) \right] \quad (16)$$

Arguing as for the derivation of Eq. 13, we get

$$\mathbb{E}_{\bar{\nu} \sim \varphi(m, \bar{b}_i)} [\Gamma(\nu_j)] \leq \frac{1}{BC} \left[m b_j^i + (B-1) \left(1 - (1 - b_j^i)^m \right) \right].$$

Then arguing as for the derivation of Eq. 14, we get

$$\sum_{m=0}^{\infty} \Pr[\nu = m] \sum_{j=1}^k \mathbb{E}_{\bar{\nu} \sim \varphi(m, \bar{b}_i)} [\Gamma(\nu_j)] \leq \frac{1}{BC} \left(\frac{1}{P_i} + (B-1) \sum_{j=1}^k \frac{p_j}{P_i + p_j} \right).$$

Substituting this into Eq. 16, we get:

$$\Pr[Y_i] \geq 1 - \frac{1}{BC} \left(\frac{1}{P_i} + (B-1) \sum_{j=1}^k \frac{p_j}{P_i + p_j} \right). \quad (17)$$

Upper bound on p_c

Substituting $\Pr[Y_i]$ from Eq. 17 in Eq. 9 we get

$$\begin{aligned}
p_c &\leq \sum_{i=1}^{k/B} P_i \frac{1}{BC} \left(\frac{1}{P_i} + (B-1) \sum_{i=1}^k \frac{p_j}{P_i + p_j} \right) \\
&= \frac{k}{B^2C} + \frac{B-1}{BC} \sum_{i=1}^{k/B} \sum_{j=1}^k \frac{P_i p_j}{P_i + p_j}.
\end{aligned}$$

□

This proves the upper bound for the equation in Theorem 1. We now prove a lower bound on that equation.

4.3.3 Lower bound

Theorem 3 When $p_i \geq 1/C$ then the expected number of cache misses in n rounds of Process “in-place” is at least $np_d + k$, where:

$$\begin{aligned} p_d \geq & \frac{1}{B} + \frac{k(2C-k)}{2C^2} + \frac{k(k-3C)}{2BC^2} - \frac{1}{2BC} - \frac{k}{2B^2C} \\ & + \frac{B(k-C) + 2C - 3k}{BC^2} \sum_{i=1}^k \sum_{j=1}^k \frac{(p_i)^2}{p_i + p_j} \\ & + \frac{(B-1)^2}{B^3C^2} \sum_{i=1}^k p_i \left[\sum_{j=1}^k \frac{p_i(1-p_i-p_j)}{(p_i+p_j)^2} - \frac{B-1}{2} \sum_{j=1}^k \sum_{l=1}^k \frac{p_i}{p_i+p_j+p_l-p_jp_l} \right] - O(e^{-B}). \end{aligned}$$

PROOF. We again consider a fixed i and consider the event X_i defined in the proof of Theorem 1. Let $\bar{\mu}$ be as defined in the proof of Theorem 1. We now obtain an upper bound on $\Pr[X_i]$.

Upper bound on $\Pr[X_i]$

In [3] it is shown that the variables μ_j are negatively associated [5]. Noting that $f(x)$ is a non-increasing function of x , then using Proposition 7 we have that:

$$\mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} \left[\prod_{j=1}^k f(\mu_j) \right] \leq \prod_{j=1}^k \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [f(\mu_j)].$$

So we can re-write Eq. 5 as:

$$\Pr[X_i] \leq \sum_{m=0}^{BC-B} \Pr[\mu = m] \prod_{j=1}^k \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [f(\mu_j)] + \sum_{m=BC-B+1}^{\infty} \Pr[\mu = m]. \quad (18)$$

We first bound the last term. We know that

$$\begin{aligned} \sum_{m=BC-B+1}^{\infty} \Pr[\mu = m] &= (1-p_i)^{BC-B+1} \sum_{m=0}^{\infty} \Pr[\mu = m] \\ &= (1-p_i)^{BC-B+1}. \end{aligned}$$

Using Proposition 5 we get that $(1-p_i)^{BC-B+1} \leq e^{-(BC-B+1)p_i}$. Assuming $p_i \geq 1/C$ the last term is at most $O(e^{-B})$.

We now bound the first term in Eq. 18. We use an approach similar to the derivation of Eq. 13 and since $\mu \leq BC-B$, so $\mu_j \leq BC-B$, we don't have to drop any terms in the simplification, so we get that:

$$\mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [f(\mu_j)] = 1 - \frac{1}{BC} (ma_j^i + (B-1)(1 - (1-a_j^i)^m)).$$

Letting $t_j(m) = \frac{1}{BC} (ma_j^i + (B-1)(1 - (1-a_j^i)^m))$ and using Proposition 4(a) we get that $e^{-\sum_j t_j(m)} \geq \prod_j (1 - t_j(m))$. So we have that

$$\begin{aligned} \Pr[X_i] &\leq \sum_{m=0}^{BC-B} \Pr[\mu = m] e^{\frac{-1}{BC} \sum_{j=1}^k (ma_j^i + (B-1)(1 - (1-a_j^i)^m))} + O(e^{-B}). \\ &\leq \sum_{m=0}^{BC-B} \Pr[\mu = m] e^{\frac{-1}{BC} (m + (B-1)(k - \sum_{j=1}^k (1-a_j^i)^m))} + O(e^{-B}). \end{aligned}$$

Using Proposition 4(b) and letting $\beta_j = (1 - a_j^i)$ we get that:

$$\begin{aligned} \Pr[X_i] \leq & \sum_{m=0}^{BC-B} \Pr[\mu = m] \left[1 - \frac{(B-1)k}{BC} - \frac{((B-1)k)^2}{2(BC)^2} - \frac{m^2}{2(BC)^2} \right. \\ & - \frac{1}{BC} \left(m - (B-1) \sum_{j=1}^k \beta_j^m \right) - \frac{(B-1)}{2(BC)^2} \left(2m \sum_{j=1}^k \beta_j^m + 2(B-1)k \sum_{j=1}^k \beta_j^m \right) \\ & \left. + \frac{(B-1)}{2(BC)^2} \left(2mk + (B-1) \sum_{j=1}^k \sum_{l=1}^k \beta_j^m \beta_l^m \right) \right] + O(e^{-B}). \end{aligned} \quad (19)$$

We now evaluate the terms in Eq. 19 assuming that $p_i \geq 1/C$, so $k \leq C$. For the simplifications of the subtractive terms we use the fact that $e^{-p_i(BC-B+1)} \leq e^{-B}$.

Since $p_i \geq 1/C$, $(1/p_i + BC - B)/(BC) = O(1)$, so using Proposition 6(a), we get that

$$\sum_{m=0}^{BC-B} \Pr[\mu = m] \frac{m}{BC} = \frac{1}{BC} \left(\frac{1}{p_i} - 1 \right) - O(e^{-B}). \quad (20)$$

Since $p_i \geq 1/C$, $(B-1)k/(BC) < 1$, so using Proposition 6(b), we get that

$$\sum_{m=0}^{\alpha-1} \Pr[\mu = m] \frac{(B-1)k}{BC} = \frac{(B-1)k}{BC} - O(e^{-B}). \quad (21)$$

We now evaluate the term

$$\begin{aligned} & \frac{(B-1)}{(BC)^2} \sum_{m=0}^{\alpha-1} \Pr[\mu = m] m \sum_{j=1}^k \beta_j^m \\ &= \frac{(B-1)}{(BC)^2} \sum_{j=1}^k \sum_{m=0}^{\infty} \Pr[\mu = m] m \beta_j^m \\ & \quad - (1-p_i)^\alpha \frac{(B-1)}{(BC)^2} \sum_{m=0}^{\infty} \Pr[\mu = m] (m + \alpha) \sum_{j=1}^k \beta_j^m \\ &= \frac{(B-1)}{(BC)^2} \sum_{j=1}^k \frac{p_i(1-p_i-p_j)}{(p_i+p_j)^2} \\ & \quad - (1-p_i)^\alpha \frac{(B-1)}{(BC)^2} \left(\sum_{j=1}^k \frac{p_i(1-p_i-p_j)}{(p_i+p_j)^2} + \sum_{j=1}^k \frac{\alpha p_i}{p_i+p_j} \right) \\ &= \frac{(B-1)}{(BC)^2} \sum_{j=1}^k \frac{p_i(1-p_i-p_j)}{(p_i+p_j)^2} - O(e^{-B}). \end{aligned} \quad (22)$$

The last simplification is due to $p_i \geq 1/C$ and $k \leq C$, so $\sum_{1 \leq j \leq k} (p_i(1-p_i-p_j))/(p_i+p_j)^2 \leq kC \leq C^2$ and $\sum_{1 \leq j \leq k} (\alpha p_i)/(p_i+p_j) \leq kCB \leq BC^2$.

Substituting back $(1 - a_j^i) = \beta_j$ and using Proposition 3 we get that

$$\begin{aligned} & \frac{(B-1)^2 k}{(BC)^2} \sum_{m=0}^{BC-B} \Pr[\mu = m] \sum_{j=1}^k \beta_j^m \\ &= \frac{(B-1)^2 k}{(BC)^2} \sum_{j=1}^k \frac{p_i}{p_i+p_j} - (1-p_i)^\alpha \frac{(B-1)^2 k}{(BC)^2} \sum_{j=1}^k \frac{p_i}{p_i+p_j} \end{aligned}$$

$$= \frac{(B-1)^2 k}{(BC)^2} \sum_{j=1}^k \frac{p_i}{p_i + p_j} - O(e^{-B}). \quad (23)$$

The last step used $\sum_{j=1}^k p_i/(p_i + p_j) \leq k$ and $((B-1)k)^2/(BC)^2 < 1$.

We now evaluate the additive terms, starting with

$$\sum_{m=0}^{BC-B} \Pr[\mu = m] \frac{mk(B-1)}{(BC)^2} \leq \frac{(B-1)k}{(BC)^2} \left(\frac{1}{p_i} - 1 \right). \quad (24)$$

Since $m < BC$ we now get that:

$$\sum_{m=0}^{BC-B} \Pr[\mu = m] \frac{m^2}{2(BC)^2} \leq \frac{1}{2BC} \left(\frac{1}{p_i} - 1 \right). \quad (25)$$

Substituting back $(1 - a_j^i) = \beta_j$ and using Proposition 3 we get that:

$$\frac{B-1}{BC} \sum_{m=0}^{BC-B} \Pr[\mu = m] \sum_{j=0}^k \beta_j^m \leq \frac{B-1}{BC} \sum_{j=1}^k \frac{p_i}{p_i + p_j}. \quad (26)$$

Finally we evaluate $\sum_{m=0}^{\alpha-1} \Pr[\mu = m] \sum_{j=1}^k \sum_{l=1}^k \beta_j^m \beta_l^m$, by first evaluating

$$\begin{aligned} (1 - p_i) \beta_j \beta_l &= (1 - p_i)(1 - a_j^i)(1 - a_l^i) \\ &= \frac{1 - 2p_i - p_j - p_l + p_i(p_i + p_j + p_l) + p_j p_l}{(1 - p_i)}. \end{aligned}$$

Using this result and Proposition 2(a), we get that

$$\begin{aligned} &\sum_{m=0}^{\alpha-1} \Pr[\mu = m] \beta_j^m \beta_l^m \\ &\leq \sum_{m=0}^{\infty} p_i \left(\frac{1 - 2p_i - p_j - p_l + p_i(p_i + p_j + p_l) + p_j p_l}{(1 - p_i)} \right)^m \\ &= \frac{p_i}{p_i + p_j + p_l - p_j p_l / (1 - p_i)} \\ &\leq \frac{p_i}{p_i + p_j + p_l - p_j p_l}. \end{aligned}$$

So we get that

$$\begin{aligned} &\frac{(B-1)^2}{2(BC)^2} \sum_{m=0}^{\alpha-1} \Pr[\mu = m] \sum_{j=1}^k \sum_{l=1}^k \beta_j^m \beta_l^m \\ &\leq \frac{(B-1)^2}{2(BC)^2} \sum_{j=1}^k \sum_{l=1}^k \frac{p_i}{p_i + p_j + p_l - p_j p_l}. \end{aligned} \quad (27)$$

Lower bound on p_d

Plugging Eqs. 20, ..., 27 into Eq. 19 we get that:

$$\begin{aligned} \Pr[X_i] &\leq 1 - \frac{1}{BC} \left(\frac{1}{p_i} - 1 \right) - \frac{(B-1)k}{BC} - \frac{(B-1)}{(BC)^2} \sum_{j=1}^k \frac{p_i(1 - p_i - p_j)}{(p_i + p_j)^2} \\ &\quad - \frac{(B-1)^2 k}{(BC)^2} \sum_{j=1}^k \frac{p_i}{p_i + p_j} + \frac{(B-1)k}{(BC)^2} \left(\frac{1}{p_i} - 1 \right) + \frac{1}{2BC} \left(\frac{1}{p_i} - 1 \right) \end{aligned}$$

$$\begin{aligned}
& + \frac{B-1}{BC} \sum_{j=1}^k \frac{p_i}{p_i + p_j} + \frac{(B-1)^2}{2(BC)^2} \sum_{j=1}^k \sum_{l=1}^k \frac{p_i}{p_i + p_j + p_l - p_j p_l} \\
& + \frac{((B-1)k)^2}{2(BC)^2} + O(e^{-B}).
\end{aligned} \tag{28}$$

Plugging Eq. 28 into Eq. 6 we get:

$$\begin{aligned}
p_d \geq \frac{1}{B} + \frac{B-1}{B} \sum_{i=1}^k p_i & \left[\left(\frac{1}{p_i} - 1 \right) \frac{1}{2BC} \left(1 - \frac{2(B-1)k}{BC} \right) \right. \\
& + \sum_{j=1}^k \frac{p_i}{p_i + p_j} \frac{B-1}{BC} \left(\frac{(B-1)k}{BC} - 1 \right) \\
& + \frac{(B-1)k}{BC} \left(1 - \frac{(B-1)k}{2BC} \right) \\
& + \frac{(B-1)}{(BC)^2} \sum_{j=1}^k \frac{p_i(1-p_i-p_j)}{(p_i+p_j)^2} \\
& \left. - \frac{(B-1)^2}{2(BC)^2} \sum_{j=1}^k \sum_{l=1}^k \frac{p_i}{p_i + p_j + p_l - p_j p_l} - O(e^{-B}) \right].
\end{aligned}$$

Simplifying further and using $\sum_{i=1}^k \sum_{j=1}^k p_i^2 / (p_i + p_j) < k$ and $\sum_{i=1}^k p_i(1/p_i - 1) = k - 1$, we get:

$$\begin{aligned}
p_d \geq \frac{1}{B} & + \frac{k(2C-k)}{2C^2} + \frac{k(k-3C)}{2BC^2} - \frac{1}{2BC} - \frac{k}{2B^2C} \\
& + \frac{B(k-C) + 2C - 3k}{BC^2} \sum_{i=1}^k \sum_{j=1}^k \frac{(p_i)^2}{p_i + p_j} \\
& + \frac{(B-1)^2}{B^3C^2} \sum_{i=1}^k p_i \left[\sum_{j=1}^k \frac{p_i(1-p_i-p_j)}{(p_i+p_j)^2} - \frac{B-1}{2} \sum_{j=1}^k \sum_{l=1}^k \frac{p_i}{p_i + p_j + p_l - p_j p_l} \right] - O(e^{-B}).
\end{aligned}$$

□

4.3.4 Upper and lower bounds for uniformly random data

Using the upper and lower bound Theorems just proven for general probability distributions, we now derive Corollaries for upper and lower bounds for uniform distribution.

Corollary 1 *If $p_1 = \dots = p_k = 1/k$ then the number of cache misses in n rounds of Process “in-place” is at most :*

$$n \left(\frac{1}{B} + \frac{k(B+5)}{2BC} + \frac{k}{B^2C} \right) + k \left(1 + \frac{1}{B} \right).$$

PROOF. Since P_i in p_c and P_j in p_d are both B/k in the equations in Theorem 2, we get that:

$$\begin{aligned}
p_d + p_c & \leq \frac{1}{B} + \frac{2(B-1)}{BC} \frac{k^2}{B} \frac{B/k}{B+1} + \frac{(B-1)^2}{B^2C} k^2 \frac{1/k}{2} + \frac{k}{B^2C} + \frac{k}{BC} \\
& = \frac{1}{B} + \frac{2(B-1)}{BC} \frac{k}{B+1} + \frac{(B-1)^2}{B^2C} \frac{k}{2} + \frac{k}{B^2C} + \frac{k}{BC} \\
& \leq \frac{1}{B} + \frac{k}{C} \left[\frac{3}{B} + \frac{B-1}{2B} \right] + \frac{k}{B^2C} \\
& = \frac{1}{B} + \frac{k(B+5)}{2BC} + \frac{k}{B^2C}.
\end{aligned}$$

□

REMARK: As we will see later, Process “in-place” models the permute phase of distribution sorting and Corollary 1 shows that one pass of uniform distribution sorting incurs $O(n/B)$ cache misses if and only if $k = O(C/B)$.

The following corollary is from the lower bound result in Theorem 3.

Corollary 2 *If $p_1 = \dots = p_k = 1/k$ then the number of cache misses in n rounds of Process “in-place” is at least:*

$$k + \frac{n}{B} + n \left[\frac{k}{2C} - \frac{k^2}{BC^2} - \frac{k+1}{2BC} - \frac{k}{2B^2C} + \frac{(B-1)^2}{12B^3C^2} (k^2(5-2B) - 7k + 2) \right].$$

PROOF. Plugging $p_i = 1/k$ in the equation in Theorem 3 we get that:

$$\begin{aligned} p_d &\geq \frac{1}{B} + \frac{k(2C-k)}{2C^2} + \frac{k(k-3C)}{2BC^2} - \frac{1}{2BC} - \frac{k}{2B^2C} \\ &\quad + \frac{B(k-C) + 2C - 3k}{BC^2} \frac{k}{2} + \frac{(B-1)^2}{B^3C^2} \left[\frac{(k-2)k}{4} - \frac{B-1}{2} \left(\frac{k^3}{3k-1} \right) \right] \\ &\geq \frac{1}{B} + \frac{k}{2C} - \frac{k^2}{BC^2} - \frac{k+1}{2BC} - \frac{k}{2B^2C} + \frac{(B-1)^2}{12B^3C^2} (k^2(5-2B) - 7k + 2). \end{aligned}$$

□

REMARK: From Corollary 1 we have that for uniformly random data and $k = \alpha C$, where $\alpha \leq 1$, other than for small values of B , the upper bound for the number of cache misses in n round is roughly

$$\frac{\alpha n}{2},$$

and from Corollary 2 we have that for uniformly random data and $k = \alpha C$, where $\alpha \leq 1$, other than for small values of B , the lower bound for the number of cache misses in n round is roughly

$$n \left(\frac{\alpha}{2} - \frac{\alpha^2}{6} \right).$$

The ratio between the upper and lower bound is $3/(3-\alpha)$. So we have that for uniformly random data the lower bound is within a factor of about $3/2$ of the upper bound when $k \leq C$ and is much closer when $k \ll C$.

4.4 Cache Analysis of Out-of-place Permutation

In this section we analyse the cache misses in a direct mapped cache during n rounds of Process “out-of-place”, introduced in Section 4.1.2. We derive a precise equation for the expected number of cache misses and closed-form upper and lower bounds. During the analysis we re-use $k, D_i, c_i, C, p_i, P_i, \bar{a}, \bar{b}, f(x)$ and $g(m)$ introduced in Section 4.3.

4.4.1 Average case analysis

We start by proving a theorem for the expected number of cache misses during n rounds of Process “out-of-place”.

Theorem 4 *The expected number X of cache misses in n rounds of Process “out-of-place” is $n(p_c + p_d + p_s) \leq X \leq n(p_c + p_d + p_s) + k(1 + 1/B) + 1$, where:*

$$p_c = \sum_{i=1}^{k/B} P_i \left(1 - \sum_{m=0}^{\infty} P_i (1 - P_i)^m \mathbb{E}_{\bar{\nu} \sim \varphi(m, \bar{b}_i)} \left[f(m+1) \prod_{j=1}^k f(\nu_j) \right] \right),$$

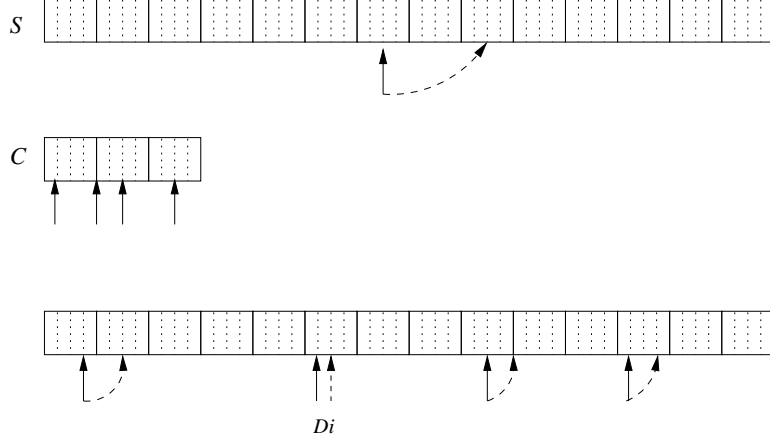


Figure 6: m rounds of Process “out-of-place”. Between two accesses to D_i , there are m accesses to “other” pointers, and $m + 1$ accesses to C , and $m + 1$ accesses to consecutive locations in S .

$$\begin{aligned}
 p_d &= \frac{1}{B} + \frac{B-1}{B} \sum_{i=1}^k p_i \\
 &\quad \left(1 - \sum_{m=0}^{\infty} p_i (1-p_i)^m E_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} \left[(1 - g(\bar{\mu})) f(m+1) \prod_{j=1}^k f(\mu_j) \right] \right), \\
 p_s &= \frac{1}{B} + \frac{B-1}{B} \left(1 - \left(1 - \frac{1}{C} \right)^2 \right).
 \end{aligned}$$

PROOF. We first analyse the miss rates for accesses to pointers D_1, \dots, D_k . Fix an i , $1 \leq i \leq k$ and a $z \geq 1$ and consider the probability of a miss between access z and $z + 1$ to pointer D_i . We define $\mu, \mu_j, m, \bar{\mu}, \bar{m}, d_{i,z}, \varphi(m, \bar{a}_i)$ and X_i as in the proof of Theorem 1. Again $f(m_j)$ is the probability that none of the m_j locations accessed by D_j in m rounds is mapped to the same cache block as location $d_{i,z}$. Similarly $g(\bar{m}) \cdot C$ is the number of count blocks accessed in m rounds, and so $1 - g(\bar{m})$ is the probability that the cache block containing $d_{i,z}$ does not conflict with the blocks from C which were accessed in these m rounds. We also have accesses to $m + 1$ contiguous locations in S and $f(m + 1)$ is the probability that these $m + 1$ accesses are not to the cache block containing $d_{i,z}$. Figure 6 shows the other memory accesses between accesses z and $z + 1$ to D_i .

For a given configuration \bar{m} of accesses, as the probabilities $f(m_j)$, $g(\bar{m})$ and $f(m + 1)$ are independent, we conclude that the probability that the cache block containing $d_{i,z}$ is not accessed in these m rounds is $(1 - g(\bar{m})) f(m) \prod_{j=1}^k f(m_j)$. Averaging over all configurations \bar{m} , we get that

$$\Pr[X_i \mid \mu = m] = E_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [(1 - g(\bar{\mu})) f(m + 1) \prod_{j=1}^k f(\mu_j)]. \quad (29)$$

Using which we get,

$$\begin{aligned}
 \Pr[X_i] &= \sum_{m=0}^{\infty} \Pr[\mu = m] \Pr[X_i \mid \mu = m] \\
 &= \sum_{m=0}^{\infty} p_i (1-p_i)^m E_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} \left[(1 - g(\bar{\mu})) f(m + 1) \prod_{j=1}^k f(\mu_j) \right].
 \end{aligned} \quad (30)$$

Arguing as for Eq. 6 we get that, other than for the first access, the probability p_d of a cache miss for a pointer access is:

$$p_d = \frac{1}{B} + \frac{B-1}{B} \sum_{i=1}^k p_i (1 - \Pr[X_i]). \quad (31)$$

Including the first access misses, the expected number of cache misses for pointer accesses is at most

$$\sum_{i=1}^k 1 + (np_i - 1) \left(\left(\frac{B-1}{B} (1 - \Pr[X_i]) \right) + \frac{1}{B} \right) \leq np_d + k. \quad (32)$$

We now consider the probability of a cache miss for an access to a count array location. Fix an $i \in \{1, \dots, k/B\}$ and a $z \geq 1$ and consider the probability of a miss between access z and $z+1$ to count block c_i . We define $\nu, \nu_j, m, \bar{\nu}, \bar{m}, P_i, \varphi(m, \bar{b}_i)$, and Y_i as in the proof of Theorem 1.

Again, given that $k \leq BC$ and assumption (c) mean that two blocks from \mathcal{C} cannot conflict with each other. So we need to determine the probability of a conflict given m_j accesses to the pointer D_j , for all $j \in \{1, \dots, k\}$, and m accesses to contiguous locations in \mathcal{S} . Again $f(m_j)$ is the probability that none of the m_j locations accessed by D_j in m rounds is mapped to the same cache block as c_i and $f(m+1)$ is the probability that the accesses to $m+1$ contiguous locations in \mathcal{S} are not to the same cache block as c_i .

So we have:

$$\begin{aligned} \Pr[Y_i] &= \sum_{m=0}^{\infty} \Pr[\nu = m] \Pr[Y_i | \nu = m] \\ &= \sum_{m=0}^{\infty} P_i (1 - P_i)^m E_{\bar{\nu} \sim \varphi(m, \bar{b}_i)} \left[f(m) \prod_{j=1}^k f(\nu_j) \right]. \end{aligned} \quad (33)$$

Arguing as for Eq. 9, the probability p_c of a cache miss for a count array access is:

$$p_c = \sum_{i=1}^{k/B} P_i (1 - \Pr[Y_i]). \quad (34)$$

Including the first access misses, the expected number of cache misses for count array accesses is at most

$$\sum_{i=1}^{k/B} 1 + (nP_i - 1)(1 - \Pr[Y_i]) \leq np_c + k/B. \quad (35)$$

We now calculate cache misses for accesses to the array \mathcal{S} . We consider the probability of a cache miss between accesses to $\mathcal{S}[s]$ and $\mathcal{S}[s+1]$. We know that there is exactly one access to a count block and one access to a pointer between two accesses to \mathcal{S} . The probability that the pointer access is to the same cache block as $\mathcal{S}[s]$ is $1/C$. The probability that a block from \mathcal{C} maps to the same cache block as $\mathcal{S}[s]$ is k/BC . Given that a block from \mathcal{C} maps to the same cache block as $\mathcal{S}[s]$, the probability that the access to the count array is to the same cache block as $\mathcal{S}[s]$ is B/k . So the probability that the pointer access is to the same cache block as $\mathcal{S}[s]$ is also $1/C$. So the probability that there are no memory accesses to the cache block that $\mathcal{S}[s]$ is mapped to before the access to $\mathcal{S}[s+1]$ is

$$(1 - 1/C)^2.$$

We have a cache miss if $\mathcal{S}[s]$ is at a cache block boundary, otherwise the probability of a cache miss is $1 - (1 - 1/C)^2$. So the probability p_s of an cache miss for an access to \mathcal{S} is

$$p_s = \frac{1}{B} + \frac{B-1}{B} \left(1 - \left(1 - \frac{1}{C} \right)^2 \right).$$

The first access to \mathcal{S} is always a cache miss, so the expected number of cache misses in accesses to \mathcal{S} is:

$$np_s + 1.$$

Plugging in the values from Eq. 30 into Eq. 32 and from Eq. 33 into Eq. 35 we get the upper bound on X , the expected number of cache misses in the processes.

The lower bound in Theorem 4 is obvious. □

4.4.2 Upper bound

We now prove a theorem on the upper bound to the expected number of cache misses during n rounds of Process “out-of-place”.

Theorem 5 *The expected number of cache misses in n rounds of Process “out-of-place” is at most $n(p_d + p_c + p_s) + k(1 + 1/B) + 1$, where:*

$$\begin{aligned} p_d &\leq \frac{1}{B} + \frac{2(B-1)k}{B^2C} + \frac{B-1}{BC} \sum_{i=1}^k \sum_{j=1}^{k/B} \frac{p_i p_j}{p_i + p_j} \\ &\quad + \frac{(B-1)^2}{B^2C} \left(1 + \sum_{i=1}^k \sum_{j=1}^k \frac{p_i p_j}{p_i + p_j} \right) \\ p_c &\leq \frac{2k}{B^2C} + \frac{B-1}{BC} \left(1 + \sum_{i=1}^{k/B} \sum_{j=1}^k \frac{p_i p_j}{p_i + p_j} \right), \\ p_s &= \frac{1}{B} + \frac{B-1}{B} \left(1 - \left(1 - \frac{1}{C} \right)^2 \right). \end{aligned}$$

PROOF. As for the upper bound for in-place permutation, in this proof we derive lower bounds for $\Pr[X_i]$ and $\Pr[Y_i]$ and we will use these to derive the upper bounds on p_d and p_c . We make extensive use of the results obtained during the proof of Theorem 1.

Again, we consider a fixed i and consider the event X_i defined in the proof of Theorem 4. We now obtain a lower bound on $\Pr[X_i]$.

Lower bound on $\Pr[X_i]$

Letting $\Gamma(x) = 1 - f(x)$ and using Proposition 1 we can rewrite Eq. 5 as:

$$\Pr[X_i] \geq \sum_{m=0}^{\infty} \Pr[\mu = m] \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} \left[1 - g(\bar{\mu}) - \Gamma(m+1) - \sum_{j=1}^k \Gamma(\mu_j) \right]. \quad (36)$$

We can use Eq. 12 as a simplification for

$$\sum_{m=0}^{\infty} \Pr[\mu = m] \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [g(\bar{\mu})],$$

and Eq. 14 as an upper bound on

$$\sum_{m=0}^{\infty} \Pr[\mu = m] \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} \left[\sum_{j=1}^k \Gamma(\mu_j) \right].$$

So we just have to evaluate

$$\sum_{m=0}^{\infty} \Pr[\mu = m] \mathbb{E}_{\bar{\mu} \sim \varphi(m, \bar{a}_i)} [\Gamma(m+1)].$$

Since we always have at least one access to \mathcal{S} , we have that

$$\begin{aligned}
\sum_{m=0}^{\infty} \Pr[\mu = m] \mathbb{E}_{\tilde{\mu} \sim \varphi(m, \bar{a}_i)}[\Gamma(\mu + 1)] &= \sum_{m=0}^{\infty} \Pr[\mu = m] \frac{m + B}{BC} \\
&\quad - \sum_{m=BC-B}^m \Pr[\mu = m] \left(\frac{m + B}{BC} - 1 \right) \\
&\leq \frac{1}{BC} \left[\sum_{m=0}^{\infty} \Pr[\mu = m] m + B \right] \\
&= \frac{1}{BC} \left(\frac{1}{p_i} - 1 + B \right), \tag{37}
\end{aligned}$$

where the last simplification used Proposition 2(c). Substituting Eq. 12, Eq. 14 and Eq. 37 in Eq. 36 we obtain the following lower bound for $\Pr[X_i]$

$$\begin{aligned}
\Pr[X_i] \geq 1 - \frac{1}{C} \sum_{j=1}^{k/B} \frac{P_j}{p_i + P_j} \\
- \frac{1}{BC} \left(\frac{2}{p_i} + (B-1) \left(1 + \sum_{j=1}^k \frac{p_j}{p_i + p_j} \right) \right). \tag{38}
\end{aligned}$$

Upper bound on p_d

Finally, substituting $\Pr[X_i]$ from Eq. 38 in Eq. 6 we get

$$\begin{aligned}
p_d &\leq \frac{1}{B} + \frac{B-1}{B} \sum_{i=1}^k p_i \\
&\quad \left(\frac{1}{C} \sum_{j=1}^{k/B} \frac{P_j}{p_i + P_j} + \frac{1}{BC} \left(\frac{2}{p_i} + (B-1) \left(1 + \sum_{j=1}^k \frac{p_j}{p_i + p_j} \right) \right) \right) \\
&= \frac{1}{B} + \frac{2(B-1)k}{B^2C} + \frac{B-1}{BC} \sum_{i=1}^k \sum_{j=1}^{k/B} \frac{p_i P_j}{p_i + P_j} \\
&\quad + \frac{(B-1)^2}{B^2C} \left(1 + \sum_{i=1}^k \sum_{j=1}^k \frac{p_i p_j}{p_i + p_j} \right).
\end{aligned}$$

We can evaluate p_c using a very similar approach to that used in the proof of Theorem 2. We again consider a fixed i and consider the event Y_i defined in the proof of Theorem 4. We now obtain a lower bound on $\Pr[Y_i]$.

Lower bound on $\Pr[Y_i]$

We can rewrite Eq. 33 as:

$$\Pr[Y_i] \geq \sum_{m=0}^{\infty} \Pr[\nu = m] \mathbb{E}_{\tilde{\nu} \sim \varphi(m, \bar{b}_i)} \left[1 - \Gamma(m+1) \sum_{j=1}^k \Gamma(\nu_j) \right]. \tag{39}$$

Eq. 17 gives us

$$\sum_{m=0}^{\infty} \Pr[\nu = m] \sum_{j=1}^k \mathbb{E}_{\tilde{\nu} \sim \varphi(m, \bar{b}_i)}[\Gamma(\nu_j)].$$

Arguing as for Eq. 37

$$\sum_{m=0}^{\infty} \Pr[\nu = m] \mathbb{E}_{\bar{\nu} \sim \varphi(m, \bar{b}_i)}[\Gamma(m+1)] \leq \frac{1}{BC} \left(\frac{1}{P_i} - 1 + B \right). \quad (40)$$

Substituting Eq. 17 and Eq. 40 in Eq. 39 we obtain the following lower bound for $\Pr[X_i]$:

$$\Pr[Y_i] \geq 1 - \frac{1}{BC} \left(\frac{2}{P_i} + (B-1) \left(1 + \sum_{j=1}^k \frac{p_j}{P_i + p_j} \right) \right). \quad (41)$$

Upper bound on p_c

Finally, substituting $\Pr[Y_i]$ from Eq. 41 in Eq. 9 we get:

$$\begin{aligned} p_c &\leq \sum_{i=1}^{k/B} P_i \frac{1}{BC} \left(\frac{2}{P_i} + (B-1) \left(1 + \sum_{j=1}^k \frac{p_j}{P_i + p_j} \right) \right) \\ &= \frac{2k}{B^2C} + \frac{B-1}{BC} \left(1 + \sum_{i=1}^{k/B} \sum_{j=1}^k \frac{P_i p_j}{P_i + p_j} \right). \end{aligned}$$

□

4.4.3 Lower bound

It is quite obvious that the lower bound for in-place permutation, given in Theorem 3, is a lower bound for out-of-place permutation.

4.4.4 Upper and lower bounds for uniformly random data

Using the upper bound Theorem just proven, we now derive a Corollary for an upper bound to the number of cache misses if the data is uniformly distributed.

Corollary 3 *If $p_1 = \dots = p_k = 1/k$ then the number of cache misses in n rounds of Process “in-place” is at most:*

$$n \left(\frac{1}{B} + \frac{k(B+3)}{2BC} + \frac{k}{B^2C} + \frac{k}{BC} \right) + k \left(1 + \frac{1}{B} \right).$$

PROOF. Since P_i in p_c and P_j in p_d are both B/k in the equations in Theorem 5, we get that

$$\begin{aligned} p_d + p_c + p_s &\leq \frac{2}{B} + \frac{2(B-1)}{BC} \frac{k^2}{B} \frac{B/k}{B+1} + \frac{(B-1)^2}{B^2C} k^2 \frac{1/k}{2} \\ &\quad + \frac{2k}{B^2C} + \frac{2(B-1)k}{B^2C} + \frac{B-1}{B} \left(1 - \frac{(C-1)^2}{C^2} \right) \\ &= \frac{2}{B} + \frac{2(B-1)}{BC} \frac{k}{B+1} + \frac{(B-1)^2}{B^2C} \frac{k}{2} \\ &\quad + \frac{2k}{B^2C} + \frac{2(B-1)k}{B^2C} + \frac{B-1}{B} \frac{2C-1}{C^2} \\ &\leq \frac{2}{B} + \frac{k}{C} \left[\frac{4}{B} + \frac{B-1}{2B} \right] + \frac{2k}{B^2C} + \frac{2}{C} \\ &= \frac{2}{B} + \frac{k(B+7)}{2BC} + \frac{2k}{B^2C} + \frac{2}{C}. \end{aligned}$$

□

REMARK: Corollaries 1 and 3 shows that for uniformly distributed data, other than for small values of B , the number of cache misses during in-place and out-of-place permutations are quite close. As for an in-place permutation, one pass of uniform distribution sorting using out-of-place permutations incurs $O(n/B)$ cache misses if and only if $k = O(C/B)$.

Using Corollary 2 for the lower bound and Corollary 3 above, we see that when $k \leq C$ the lower bound is again within $3/2$ of the upper bound and is much closer when $k \ll C$.

4.5 Cache Analysis of Multiple Sequences Access

Accessing k sequences is like Process “in-place” in Section 4.1.1 except that there is no interaction with a count array, so we delete step 2 and assumption (c). An analogue of Theorem 1 is easily obtained. An easy modification to the proof of Theorem 2 gives:

Theorem 6 *The expected number of cache misses in n rounds of sequence accesses is at most:*

$$k + n \left(\frac{1}{B} + \frac{k(B-1)}{B^2C} + \frac{(B-1)^2}{B^2C} \sum_{i=1}^k \sum_{j=1}^k \frac{p_i p_j}{p_i + p_j} \right).$$

Corollary 4 *If $p_1 = \dots = p_k = 1/k$ then the number of cache misses in n rounds of sequence accesses is at most:*

$$n \left(\frac{1}{B} + \frac{k(B+3)}{2BC} \right) + k.$$

REMARK: From Corollary 4, $k = O(C/B)$ random sequences can be accessed incurring an optimal $O(n/B)$ misses. This essentially agrees with the results obtained by Mehlhorn and Sanders [9] and Sen and Chatterjee [14].

REMARK: Since its derivation ignored the effects of the count array, the lower bound in Theorem 3 applies directly to sequence accesses. Note that the lower bound we obtain for uniformly random data, as stated in Corollary 2, is sharper than the lower bound of $0.25(1 - e^{-0.25k/C})$ obtained in [14].

REMARK: Our upper and lower bounds are also closer than those in [9]. The analysis in [9] assumes that accesses to the sequences are controlled by an adversary; our analysis demonstrates, that with uniform randomised accesses to the sequences, more sequences can be accessed optimally.

4.6 Correspondence between the processes and the permute phase

We now show how the Processes “in-place” and “out-of-place” model the permute phase of a generic distribution sorting algorithm.

The correspondence between Process “in-place” of Section 4.1.1 and the pseudocode in Figure 2 is as follows. Each iteration of the inner loop (steps 3.1-3.5) of the pseudocode corresponds to a round of Process “in-place”. The array `COUNT` corresponds to the locations \mathcal{C} , and the pointer D_i points to `DATA[idx]`. The variables x in the process and the pseudocode play a similar role. It can easily be verified that in each iteration of the loop in the pseudocode, the value of x is any integer $1, \dots, k$ with probability p_1, \dots, p_k , independently of its previous values, as in Step 1 of Process “in-place”. A read at a location immediately followed by a write to the same location is counted as one access. Thus, the read and increment of `COUNT[x]` in Steps 3.2 and 3.3 of the pseudocode constitutes one access, equivalent to Step 2. Similarly the “swap” in Step 3.5 of the pseudocode corresponds to the memory access in Step 3 of the process. The process does not model the initial access in Step 1 of the pseudocode, and nor does it model the task of looking for new cycle leaders in Steps 4 and 5 of the pseudocode.

The correspondence between Process “out-of-place” of Section 4.1.2 and the pseudocode in Figure 1 is as follows. The array `COUNT` corresponds to the locations \mathcal{C} , the array `DATA` corresponds to the locations \mathcal{S} , i in the pseudocode corresponds to s in the process, and the pointer D_i points to `DEST[idx]`. The increment of i in the pseudocode is equivalent to the increment of s in the process,

and the accesses to $\text{DATA}[i]$ and $\mathcal{S}[s]$ are equivalent. As above, the variables x in the process and the pseudocode in the pseudocode play a similar role. Again, the read and increment of $\text{COUNT}[x]$ of the pseudocode constitutes one access, and is equivalent to Step 3 of the process. The access to $\text{DEST}[idx]$ of the pseudocode corresponds to the memory access in Step 4 of the process.

Assumption (b) of the processes is clearly satisfied and assumption (c) can normally be made to hold. Assumption (d) and $k \leq CB$ or $k \leq C$ may not hold in practice, in [11] we give an approximate analysis which deals with this. Assumption (a) of the processes, that the starting locations of the pointers D_i are uniformly and independently distributed, is patently false, we discuss this in more detail in [11]. We may force it to hold by adding random offsets to the starting location of each pointer, at the cost of needing more memory and adding a compaction phase after the permute, this has also been suggested by Mehlhorn and Sanders [9]. This only works if the permute is not in-place, and if k is sufficiently small (e.g. $k \leq n/(CB)$). In [11] we study assumption (a) empirically in the context of uniform distribution sorting. Another weakness is that our processes are continuous, so the sequence lengths are not specified, whereas in distribution sorting we sort n keys and each sequence is of a finite length.

5 MSB radix sort

We now consider the problem of sorting n independent and uniformly-distributed floating-point numbers in the range $[0, 1)$ using the integer sorting algorithm MSB radix sort. As noted earlier, it suffices to sort lexicographically the bit-strings which represent the floats, by viewing them as integers. One pass of MSB radix sort using radix size r groups the keys according to their most significant r bits in $O(2^r + n)$ time. For random integers, a reasonable choice for minimising instruction counts is $r = \lceil \log n - 3 \rceil$ bits, or classifying into about $n/8$ classes. Since each class has about 8 keys on average, they can be sorted using insertion sort. Using this approach for this problem gives terrible performance even at small values of n (see Table 1). As we now show, the problem lies with the distribution of the integers on which MSB radix sort is applied.

5.1 Radix sorting floating-point numbers

A floating-point number is represented as a triple of non-negative integers $\langle i, j, l \rangle$. Here i is called the *sign bit* and is a 0-1 value (0 indicating non-negative numbers, 1 indicating negative numbers), j is called the *exponent* and is represented using e bits and l is called the *mantissa* and represented using m bits. Let $j^* = j - 2^{e-1} + 1$ denote the *unbiased* exponent of $\langle i, j, l \rangle$. Roughly following the IEEE 754 standard, let the triple $\langle 0, 0, 0 \rangle$ represent the number 0, and let $\langle i, j, l \rangle$, where $j > 0$, represent the number $\pm 2^{j^*} (1 + l2^{-m})$, depending on whether $x = 0$ or 1; no other triple is a floating-point number. Internally each member of the triple is stored in consecutive fields of a word. The IEEE 754 standard specifies $e = 8$ and $m = 23$ for 32-bit floats and $e = 11$ and $m = 52$ for 64-bit floats [4].

We model the generation of a random float in the range $[0, 1)$ as follows: generate an (infinite-precision) random real number, and round it down to the next smaller float. On average, half the numbers generated will lie in the range $[0.5, 1)$ and will have an unbiased exponent of -1 . In general, for all non-zero numbers, the unbiased exponent has value i with probability 2^i , for $i = -1, -2, \dots, -2^{e-1} + 2$, whereas the mantissa is a random m -bit integer. The value 0 has probability $2^{-2^{e-1}+2}$. Clearly, the distribution is not uniform, and it is easy to see that the average size of the largest class after the first pass of MSB radix sort with radix r is $n \left(1 - \frac{1}{2^{2^e-r+1}}\right)$ if $r < e + 1$, and $n/(2^{r-e})$ if $r \geq e + 1$.

This shows, e.g., that the largest sub-problems in the examples of Table 1 would be of size $n/2^{\lceil \log n - 3 \rceil - 11} \approx 2^{14}$, so using insertion sort after one pass is inefficient in this case². To get down to problems of size 8 in one pass requires a radix of about $\log n + 8$, which is impractical. Also, MSB radix sort applied to random integers has $O(n)$ expected running time independently of the word size, but this is not true for floats. A first pass with $r \ll e$ barely reduces the largest problem

²In fact, the total number of keys in all sub-problems of this size would be $n/2$ on average.

size, and the same holds for subsequent passes until bits from the mantissa are reached. As the radix in any pass is limited to $\log n + O(1)$ bits, we may need $\Omega(e/\log n)$ passes, introducing a dependence on the word size.

5.2 Using Quicksort

To get around the problem of having several passes before we reduce the largest class, we partition the input keys around a value $1/n \leq \theta \leq 1/(\log n)$, and sort the keys smaller than θ in $O(n)$ expected time using Quicksort. We then apply MSB radix sort to the remaining keys. Let $e' = \min\{\lceil \log \log(1/\theta) \rceil, e\}$ denote the *effective exponent*, since the remaining keys have exponents which vary only in the lower order e' bits. This means that keys can be grouped according to a radix $r = e + 1 + m'$ with $m' \geq 0$ in $O(n + 2^{e'+m'})$ time and $O(2^{e'+m'})$ space. Since $e' = O(\log \log n)$, we can take up to $\log n - O(\log \log n)$ bits from the mantissa as part of the first radix; as all sub-problems now only deal with bits from the mantissa they can be solved in linear expected time, giving a linear running time overall.

5.3 Cache analysis

We now use our analysis to calculate an upper bound for the cache misses in the permute phase of the first pass of MSB radix sort using a radix $r = e + 1 + m'$, for some $m' \geq 0$, assuming also that all keys are in the range $[\theta, 1)$, for some $\theta \geq 1/n$. There are $2^{e'+m'}$ pointers in all, which can be divided into $g = 2^{e'}$ groups of $K = 2^{m'}$ pointers each. Group i corresponds to keys with unbiased exponent $-i$, for $i = 1, \dots, g$. All pointers in group i have an access probability of $1/(K2^i)$. Using Theorem 1 and a slight extension of the methods of Theorem 2 we are able to prove Theorem 7 below, which states that the number of misses is essentially independent of g :

Theorem 7 *Provided $gK \leq CB$ and $K \leq C$ the number of misses in the first pass of the permute phase of MSB radix sort is at most:*

$$n \left(\frac{1}{B} + \frac{2K}{BC} (2.3B + 2 \log B + \log C - \log K + 0.7) \right) + gK(1 + 1/B).$$

PROOF. Using Eq. 15 we can calculate an upper bound on the probability of event $X_{(i-1)K+1}$ as:

$$\begin{aligned} \Pr[X_{(i-1)K+1}] &\geq 1 - \frac{K2^i}{BC} - \frac{1}{C} \sum_{j=1}^g \sum_{l=1}^{K/B} \frac{B2^{-j}}{B2^{-j} + 2^{-i}} - \frac{B-1}{BC} \sum_{j=1}^g \sum_{l=1}^K \frac{2^{-j}}{2^{-j} + 2^{-i}} \\ &= 1 - \frac{K}{BC} \left(\sum_{j=1}^g \frac{B2^{-j}}{B2^{-j} + 2^{-i}} + 2^i + (B-1) \sum_{j=1}^g \frac{2^{-j}}{2^{-j} + 2^{-i}} \right) \\ &\geq 1 - \frac{K}{BC} \left(\log B + i + 2^i + (B-1) \sum_{j=1}^g \frac{2^{-j}}{2^{-j} + 2^{-i}} \right). \end{aligned} \tag{42}$$

If $K2^i/(BC) \geq 1$ then $\Pr[X_{(i-1)K+1}]$ would be negative, so we place a bound on this term such that $K2^i < BC$. The maximum value of i such that $K2^i/(BC) < 1$ is $\log BC - \log K - 1$.

Since the probabilities of access to pointer $D_{(i-1)K+1}, \dots, D_{iK}$ are all $1/(K2^i)$ we can calculate an upper bound on p_d using Eq. 6 and 42 as:

$$p_d \leq \frac{K^2}{BC} \left(\sum_{i=1}^g p_i \left(\log B + i + (B-1) \sum_{j=1}^g \frac{2^{-j}}{2^{-j} + 2^{-i}} \right) + \sum_{i=1}^{\log BC - \log K - 1} p_i 2^i \right)$$

$n =$	1×10^6	2×10^6	4×10^6	8×10^6	16×10^6	32×10^6
MTQuick	0.7400	1.5890	3.3690	7.2430	15.298	32.092
Naive1	7.0620	14.192	28.436	57.082	115.16	233.16

Table 1: Memory-tuned Quicksort and Naive1 MSBRadix. Running times in seconds of memory-tuned Quicksort and Naive1 MSBRadix sort (single pass MSBRadix sort without partitioning, $r = \lceil \log n - 3 \rceil$) floating point keys.

$$\begin{aligned}
& + \sum_{i=\log BC - \log K}^g K p_i \\
= & \sum_{i=1}^g \frac{1}{2^i} \frac{K}{BC} \left(\log B + i + (B-1) \sum_{j=1}^g \frac{2^{-j}}{2^{-j} + 2^{-i}} \right) \\
& + \sum_{i=1}^{\log BC - \log K - 1} \frac{1}{2^i} \frac{K}{BC} 2^i + \sum_{i=\log BC - \log K}^g \frac{1}{2^i} \\
\leq & \frac{K}{BC} \left(\sum_{i=1}^g \frac{\log B}{2^i} + \sum_{i=1}^g \frac{i}{2^i} + (B-1) \sum_{i=1}^g \frac{1}{2^i} \sum_{j=1}^g \frac{2^{-j}}{2^{-j} + 2^{-i}} \right) \\
& + \frac{K}{BC} (\log BC - \log K - 1) + \frac{2K}{CB} \\
\leq & \frac{K}{BC} (2 \log B + 3 + \log C - \log K + 2.3(B-1)).
\end{aligned}$$

□

5.4 Tuning MSB radix sort

We now optimise parameter choices in our algorithms. The smaller the value of θ , the fewer keys are sorted by Quicksort, but reducing θ may increase e' . A larger value of e' does not mean more misses, by Theorem 7, but it does mean a larger count array. We choose $\theta = 1/(\log n)^2$ as a compromise, ensuring that Quicksort uses $o(n)$ time. Using the above analysis we are also able to determine an optimal number of classes to use in each sorting sub-problem. We use two criteria of optimality. In the first, we require that each pass incur no more than $(2 + \varepsilon)n/B$ misses for some constant $\varepsilon > 0$, thus seeking essentially to minimise cache misses ($2n/B$ misses is the bare minimum for the count and permute phases). In the second, we trade-off reductions in cache misses against extra computation. The latter yields better practical results, and results shown below are for this approach.

5.5 Experimental results

Table 2 compares tuned MSB radix sort with memory-tuned Quicksort[8] and MPFlashsort [11], a memory-tuned version of a distribution sorting algorithm which assumes that the keys are independently drawn from a uniformly random distribution. The algorithms were coded in C and compiled using gcc 2.8.1. The experiments were our Sun UltraSparc-II with 2×300 MHz processors and 1GB main memory, and a 16KB L1 data cache, 512KB L2 direct-mapped cache. Observe that MSB radix sort easily outperforms the other algorithms for the range of values considered.

6 Conclusions

We have analysed the average-case cache performance of the permute phase of distribution sorting when the keys are independently but not uniformly distributed. We have presented equations for

$n =$	1×10^6	2×10^6	4×10^6	8×10^6	16×10^6	32×10^6	64×10^6
MPFlash	0.6780	1.3780	2.2756	6.1700	13.308	27.738	56.796
MTQuick	0.7400	1.5890	3.3690	7.2430	15.298	32.092	67.861
MSBRadix	0.3865	0.8470	1.9820	5.0300	9.4800	19.436	40.663

Table 2: MPFlashsort, memory-tuned Quicksort and MSBRadix. Running times in seconds of MPFlashsort, memory-tuned Quicksort and MSBRadix sort on a Sun UltraSparc-II using single precision floating point keys.

the number of misses during in-place and out-of-place permutations and have given closed-form upper and lower bounds on these. We have shown that the upper and lower bounds are quite close when $k \leq C$ and the data is known to be independently and uniformly distributed. We have shown how this analysis can easily be extended to obtain the number of cache misses during accesses to multiple sequences.

We have shown that if the integer sorting algorithm MSB radix sort is used to sort uniformly and randomly distributed floating point numbers then a non-uniform distribution of keys to classes is induced. We have shown that a naive implementation of this algorithm would have very poor performance due to this non-uniform distribution. We have shown that by partitioning the keys, to remove keys which are expected to go into small classes, and by using our analysis, the algorithm can be tuned for good cache performance. Due to fast integer operations and good cache utilisation the tuned algorithm outperforms MPFlashsort, a cache-tuned distribution sorting algorithm, and memory-tuned Quicksort.

References

- [1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [2] L. Arge. External memory data structures (Invited Paper). In *Proc. 9th Annual European Symposium on Algorithms*, LNCS 2161, pp. 1–29, 2001.
- [3] D. Dubhashi and D. Ranjan. Balls and Bins: A Study in Negative Dependence. *Random Structures and Algorithms* **13**, pp. 99–124, 1998.
- [4] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, 2nd ed.*. Morgan Kaufmann, 1996.
- [5] K. Joag-Dev and F. Proschan. Negative association of random variables, with applications. *Annals of Statistics* **11**, pp. 286–295, 1983.
- [6] D. E. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching, 3rd ed.*. Addison-Wesley, 1997.
- [7] R. E. Ladner, J. D. Fix and A. LaMarca. Cache Performance Analysis of Traversals and Random Accesses. In *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 613–622, 1999.
- [8] A. LaMarca and R. E. Ladner. The influence of caches on the performance of sorting. *Journal of Algorithms* **31**, pp. 66–104, 1999.
- [9] K. Mehlhorn and P. Sanders. Scanning Multiple Sequences via Cache Memory. *Algorithmica* **35(1)**: pp. 75–93, 2003. Preliminary version in *Proc. 3rd Workshop on Algorithm Engineering*, Preliminary version in *Proc. 26th Annual International Colloquium on Automata, Languages and Programming*, LNCS 1555, pp. 655–664, 1999.
- [10] N. Rahman. Internal Memory Sorting and Searching. Ph.D. Thesis. King’s College, University of London.

- [11] N. Rahman and R. Raman. Analysing Cache Effects in Distribution Sorting. *ACM Journal of Experimental Algorithmics* **5**, Article 14, 2001. Preliminary version in *Proc. 3rd Workshop on Algorithm Engineering*, LNCS 1668, pp. 184–198, 1999.
- [12] N. Rahman and R. Raman. Adapting radix sort to the memory hierarchy. *ACM Journal of Experimental Algorithmics*, (to appear). Preliminary version in *Proc. 2nd Workshop on Algorithm Engineering and Experiments*, 2000.
- [13] N. Rahman and R. Raman. Analysing the Cache Behaviour of Non-uniform Distribution Sorting Algorithms. In *Proc. 8th Annual European Symposium on Algorithms*, LNCS 1879, pp. 380–391, 2000.
- [14] S. Sen and S. Chatterjee. Towards a theory of cache-efficient algorithms (extended abstract). In *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 829–838, 2000.
- [15] J. S. Vitter. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Computing Surveys* **33**, pp. 209–271, 2001.